CSCI 410 Pattern Recognition

Multi-layer Perceptron

by Denton Bobeldyk

In this lab, you will be using Matlab to create a multi-layer perceptron with 3 layers: input layer, hidden layer, output layer (using a sigmoid function). Please follow the steps below:

1. Create a function titled ‘week4Solution<yourLastName>’. Where <yourLastName> is your last name.

2. Define the learning rate and total iterations

   ```
   learningRate = 0.5;

   totalIterations = 500;
   ```

3. Define the size of the input layer and the hidden layer:

   ```
   inputLayerNumber = 2;

   hiddenLayerNumber = 2;
   ```

4. Define the input and hidden layer:

   ```
   inputLayer = zeros(inputLayerNumber, 1);
   hiddenLayer = zeros(hiddenLayerNumber, 1);
   ```

5. Define the output layer:

   ```
   outputLayer = 0;
   ```

6. Randomly assign the weights to the input and hidden layer:

   ```
   inputLayerWeights = rand( (inputLayerNumber + 1) ,hiddenLayerNumber) - .5 ;

   hiddenLayerWeights = rand( (hiddenLayerNumber + 1), 1) - .5;
   ```

7. Define the input data:

   ```
   inputLayer = [0 0; 0 1; 1 0; 1 1];
   ```

8. Define the target output for the input layer:

   ```
   ANDtargetOutput = [0; 0; 0; 1];
   targetOutput = ANDtargetOutput;
   ```

9. Define the variable `m’ as the number of samples:

   ```
   m = size(targetOutput, 1);
   ```

10. Add the bias to the input and hidden layer:
    inputLayerWithBias = [ones(m,1) inputLayer];
    hiddenLayerWithBias = zeros(hiddenLayerNumber + 1, 1);

11. Create a for loop, that will step through each of the samples one at a time (Note: this is known as online learning)

```
      for iter=1:totalIterations

       for i = 1:m

        hiddenLayerActivation = inputLayerWithBias(i, :) * inputLayerWeights;
        hiddenLayer = sigmoid(hiddenLayerActivation);

        %Add the bias to the hiddenLayer
        hiddenLayerWithBias = [1, hiddenLayer];


        outputLayer = sigmoid(hiddenLayerWithBias * hiddenLayerWeights);

      %Calculate the error:
      deltaOutput = targetOutput(i) - outputLayer;
              deltaHidden(1) = (deltaOutput * hiddenLayerWeights(1)) .*
      ((hiddenLayerWithBias(1) * (1.0 - hiddenLayerWithBias(1))));
              deltaHidden(2) = (deltaOutput * hiddenLayerWeights(2)) .*
      ((hiddenLayerWithBias(2) * (1.0 - hiddenLayerWithBias(2))));
              deltaHidden(3) = (deltaOutput * hiddenLayerWeights(3)) .*
      ((hiddenLayerWithBias(3) * (1.0 - hiddenLayerWithBias(3))));

         % Fixed Step Gradient Descent - Update the weights
        hiddenLayerWeights(1) = hiddenLayerWeights(1) + (learningRate *
(deltaOutput * hiddenLayerWithBias(1)));
        hiddenLayerWeights(2) = hiddenLayerWeights(2) + (learningRate *
(deltaOutput * hiddenLayerWithBias(2)));
        hiddenLayerWeights(3) = hiddenLayerWeights(3) + (learningRate *
(deltaOutput * hiddenLayerWithBias(3)));

        %update each weight according to the part that they played
        inputLayerWeights(1,1) = inputLayerWeights(1,1) + (learningRate *
deltaHidden(2) * inputLayerWithBias(i, 1));
        inputLayerWeights(1,2) = inputLayerWeights(1,2) + (learningRate *
deltaHidden(3) * inputLayerWithBias(i, 1));

        inputLayerWeights(2,1) = inputLayerWeights(2,1) + (learningRate *
deltaHidden(2) * inputLayerWithBias(i, 2));
        inputLayerWeights(2,2) = inputLayerWeights(2,2) + (learningRate *
deltaHidden(3) * inputLayerWithBias(i, 2));

        inputLayerWeights(3,1) = inputLayerWeights(3,1) + (learningRate *
deltaHidden(2) * inputLayerWithBias(i, 3));
        inputLayerWeights(3,2) = inputLayerWeights(3,2) + (learningRate *
deltaHidden(3) * inputLayerWithBias(i, 3));
```

```
        end

    end
```

12. Create a sigmoid function:

```
    function a = sigmoid(z)

    a = 1.0 ./ (1.0 + exp(-z));

    end
```

13. Create a cost function:

```
% This function will only work for NN with just one output (k = 1)
function [averageCost] = costFunction(inputLayerWithBias,
inputLayerWeights, hiddenLayerWeights, targetOutput)
%Sum of square errors cost function
m = 4;
hiddenLayer = sigmoid(inputLayerWithBias * inputLayerWeights);

hiddenLayerWithBias = [ones(m,1) hiddenLayer];

outputLayer = sigmoid(hiddenLayerWithBias * hiddenLayerWeights);


% Step through all of the samples and calculate the cost at each one
for i=1:m
    cost(i) = (1/2) * ((outputLayer(i) - targetOutput(i)) .^ 2);
end

%Sum up all of the individual costs
totalCost = sum(cost);

%average them out
averageCost = totalCost * (1/m);


end
```

14. Create a function that will summarize the output of the 4 samples:

```
function outputSummary(inputLayerWithBias, inputLayerWeights,
hiddenLayerWeights, targetOutput, totalIterations)

cost = costFunction(inputLayerWithBias, inputLayerWeights,
hiddenLayerWeights, targetOutput);
```

```
hiddenLayer = sigmoid(inputLayerWithBias * inputLayerWeights);

%we have multiple samples, so we need to add the bias to each of them
hiddenLayerWithBias = [ones(size(targetOutput,1),1) hiddenLayer];

actualOutput = sigmoid(hiddenLayerWithBias * hiddenLayerWeights);



fprintf('\n\n========================================\n');
fprintf('Output Summary (after %d iterations):\n', totalIterations);
fprintf('Total Cost: [%f]\n', cost);


for i=1:length(actualOutput)
    if(actualOutput(i) > 0.5)
        thresholdedValue = 1;
    else
        thresholdedValue = 0;
    end

    if(thresholdedValue == targetOutput(i))
        fprintf('Sample[%d]: Target = [%f] Thresholded Value = [%f]
Actual= [%f]\n', i, targetOutput(i), thresholdedValue, actualOutput(i));
    else  % else print the error in red
        fprintf(2,'Sample[%d]: Target = [%f] Thresholded Value = [%f]
Actual= [%f]\n', i, targetOutput(i), thresholdedValue, actualOutput(i));
    end


end

fprintf('========================================\n\n\n');

end
```

15. Insert a line containing a call to ‘`outputSummary`’ in the appropriate place. This line should output the summary of the trained network.
16. Attempt to learn the following target outputs by setting the ‘targetOutput’ variable equal to each of the below:

```
ANDtargetOutput = [0; 0; 0; 1];
ORtargetOutput = [0; 1; 1; 1];
NANDtargetOutput = [1; 1; 1; 0];
NORtargetOutput = [1; 0; 0; 0];
XORtargetOutput = [0; 1; 1; 0];
```

17. Which of the above target outputs does the network have the hardest time learning and explain why?

18. Insert the above answer into your script using ‘fprintf’ statement(s).

Turn-in:

1. A single matlab script that performs the above tasks (Please no zip files)
2. A word document containing the copy and pasted output of the script execution. Please make sure there are no line wrappings (decrease the font if necessary).

References:

Andrew NG, Machine Learning course from Coursera

Bishop, Christopher M. *Neural networks for pattern recognition*. Oxford university press, 1995.